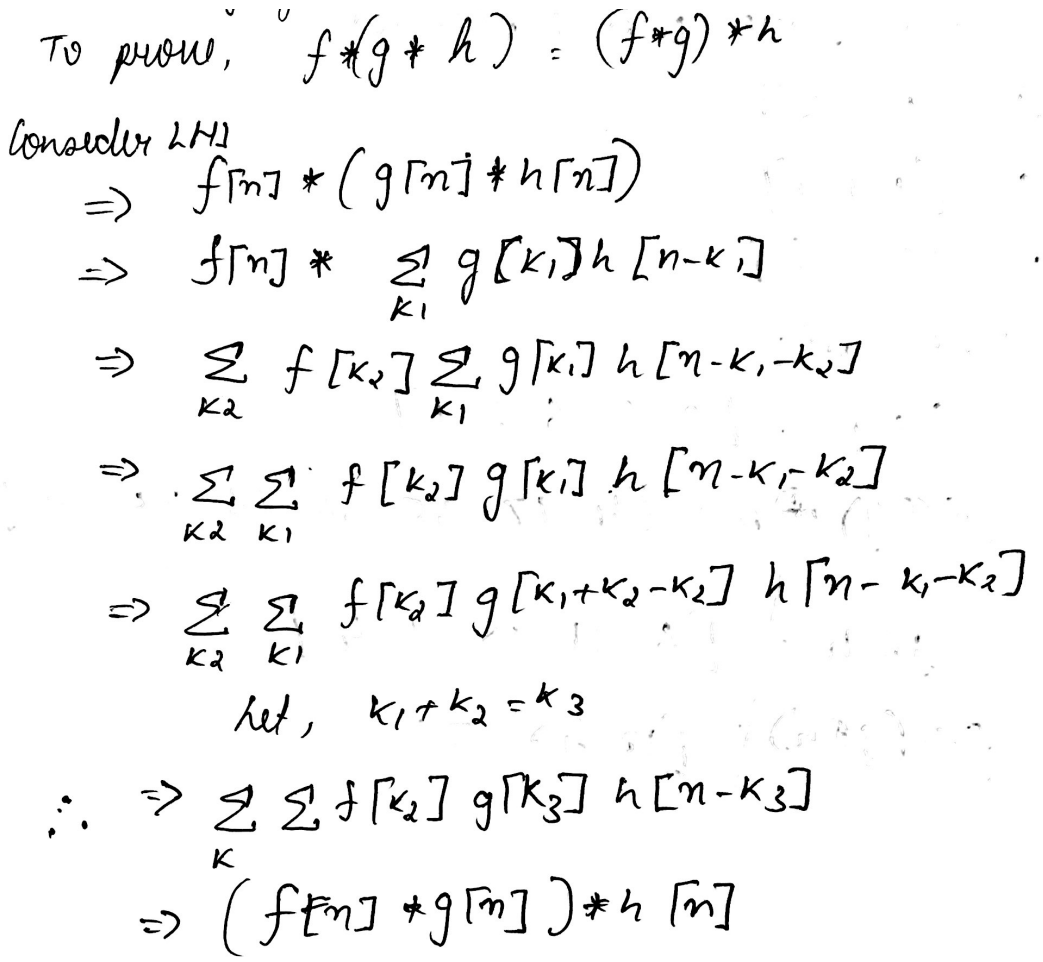
Image Understanding – Homework 2

Kishore Narendran – narendrk@uci.edu - 14644574

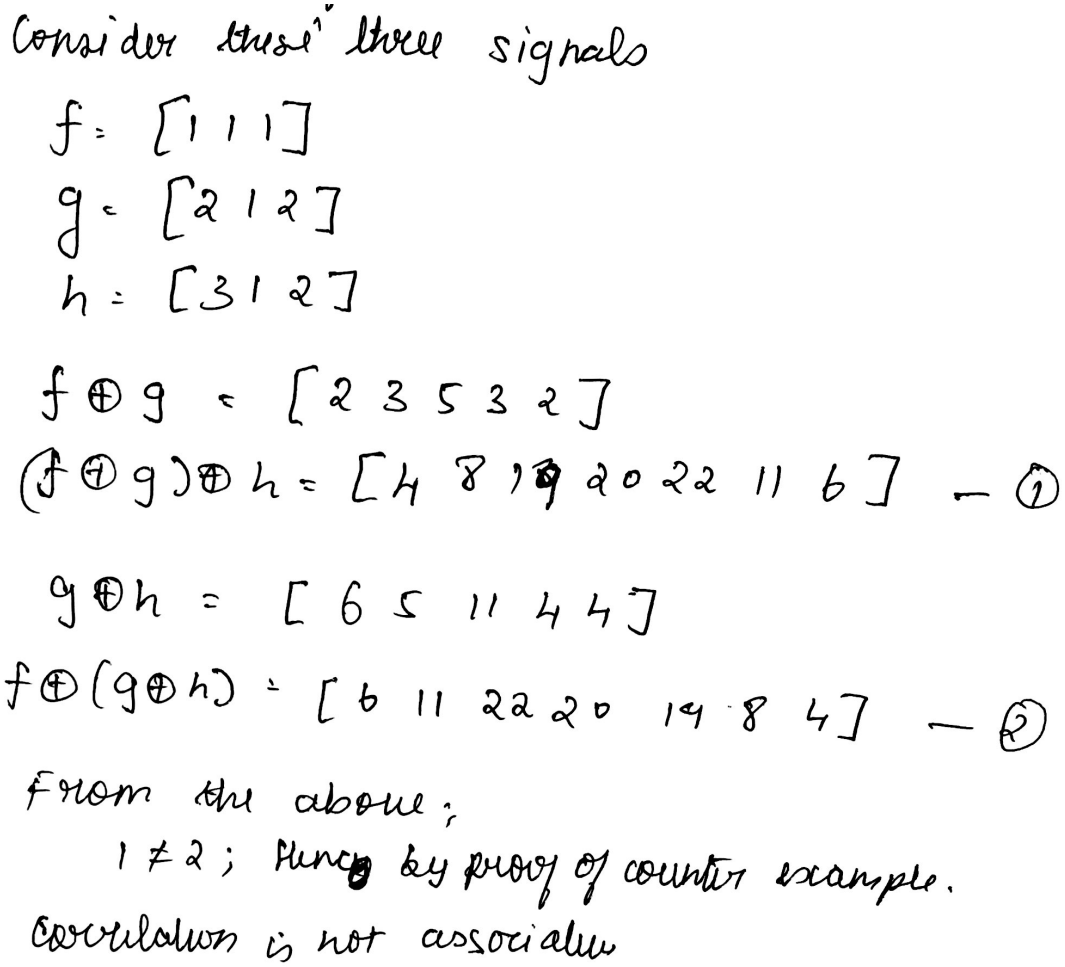
1) Associativity of Convolution

The associativity of convolution has been proven as shown in the following image.



2) Non-Associativity of Correlation

Correlation is not associative in nature, and it has been proven by counter example in the following image.



3) Equivalence of 2D Isotropic Gaussian and two 1D Gaussian Signals

The equivalence is proven, as shown in the following image.

$$g_1(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \cdot e^{-x^2/2\sigma_1^2}$$

$$g_2(y) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \cdot e^{-y^2/2\sigma_2^2}$$

$$g_1(x) \cdot g_1(y) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \cdot e^{\frac{-x^2}{2\sigma_1^2} + \frac{-y^2}{2\sigma_2^2}}$$

Consider $\sigma_1 = \sigma_2 = \sigma$

Then,

$$g_1(x)\, g_2(y) = \frac{1}{2\sigma^2 \cdot \pi} \cdot e^{-(x^2+y^2)/2\sigma^2} \quad \text{— (1)}$$

2D isotropic gaussian function is,

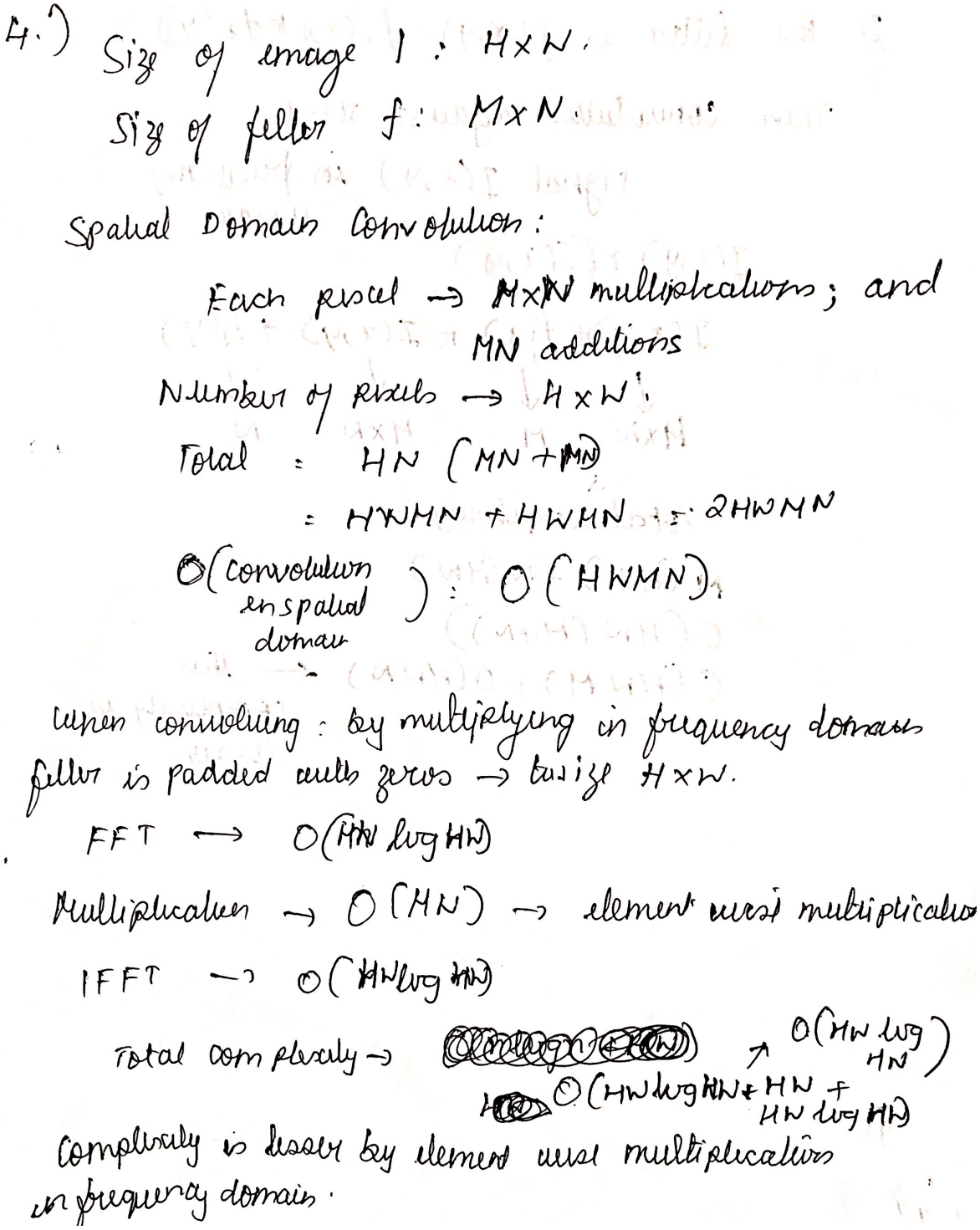
$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-(x^2+y^2)/2\sigma^2} \quad \text{— (2)}$$

From 1 = 2 ; Hence proved

$$g(x,y) = g_1(x) \times g_2(y)$$

4) Complexities of Convolution of Signals in Spatial Domain vs Multiplication of Signals in Frequency Domain

The complexities of forming convolution of two signals in spatial domain, vs multiplication of the signals in frequency domain is shown in the following image.



If the filter can be expressed as a product of two 1D filters. Then the convolution with a signal in the frequency domain can be done in the following complexity.

If the filter is $f(x,y) = f_1(x) * f_2(y)$

Then convolution against some signal $I(x,y)$ in frequency domain

$I(x,y) * f(x,y)$

$I(x,y) * f_1(x) + I(x,y) * f_2(y)$

$H \times W$ $\quad$ M $\quad$ $H \times N$ $\quad$ N

Total complexity is

$M(HN) + N(HW)$

$O(HN(M+N))$

$O(HNM) + O(HNN)$ ← this complexity is linear.

5) Variance of the convolution of two Gaussian Signals

Consider 2 gaussian signals:

$$g_1(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \cdot e^{-x^2/2\sigma_1^2}$$

$$g_2(x) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \cdot e^{-x^2/2\sigma_2^2}$$

The gaussians in frequency domain are also gaussians

$$G_1(\omega) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{\frac{-\sigma_1^2\omega^2}{2}} \qquad G_2(\omega) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \cdot e^{\frac{-\sigma_2^2\omega^2}{2}}$$

Convolution in spatial domain is multiplication in frequency domain

$$g_1(x) * g_2(x) \Rightarrow G_1(\omega) \cdot G_2(\omega)$$

$$= \frac{1}{2\pi\sigma_1^2\sigma_2^2} \cdot e^{-\omega^2 \cdot \left(\frac{\sigma_1^2+\sigma_2^2}{2}\right)}$$

Inverse DFT of Gaussian is also a Gaussian

$$g_3(x) = IDFT\left(\frac{1}{2\pi\sigma_1^2\sigma_2^2} \cdot e^{-\omega^2\left(\frac{\sigma_1^2+\sigma_2^2}{2}\right)}\right)$$

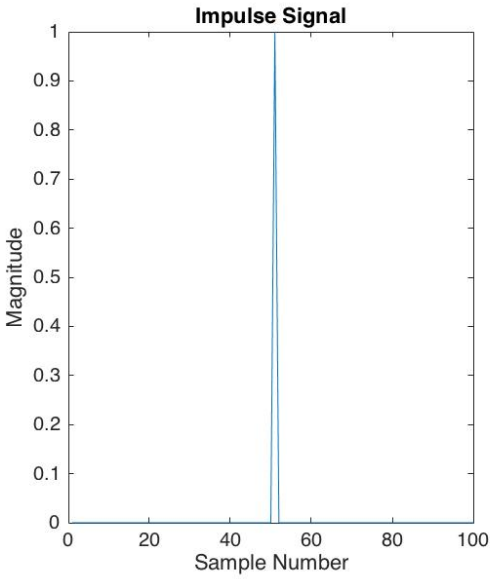
$$= \frac{1}{2\pi\sigma_1\sigma_2} \cdot e^{-x^2/2(\sigma_1^2+\sigma_2^2)} \quad \leftarrow \text{Gaussian}$$

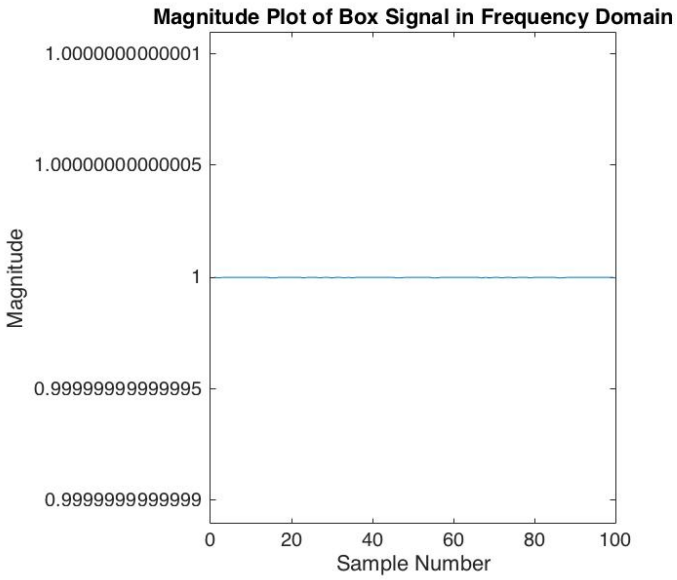
$$\sigma_3^2 = \sigma_1^2 + \sigma_2^2$$

This is useful in practice because computations are less expensive in frequency domain. Decomposition of gaussians allows for simpler computations by multiplications in frequency domain

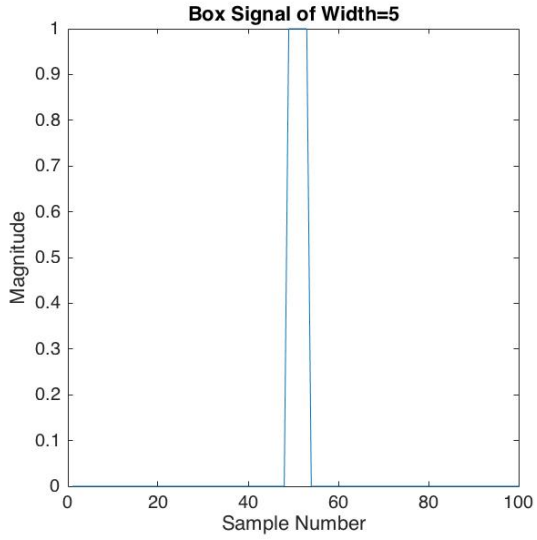
6) Experiments with DFT and Some Signals

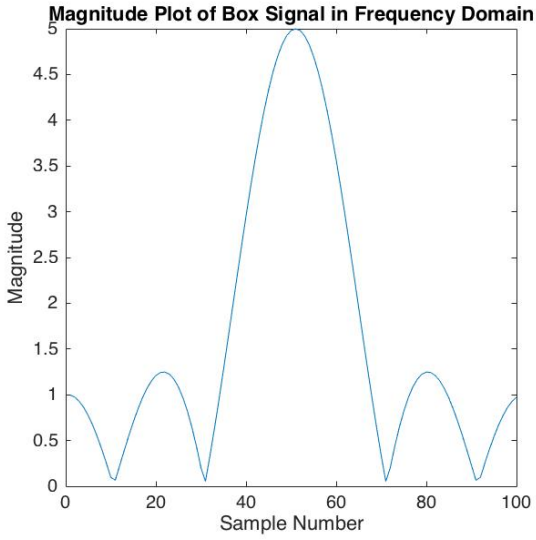
An impulse signal, and its magnitude plot in the frequency domain.



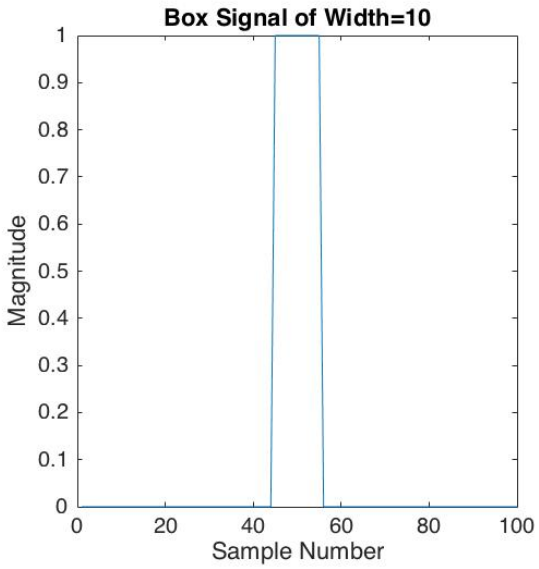


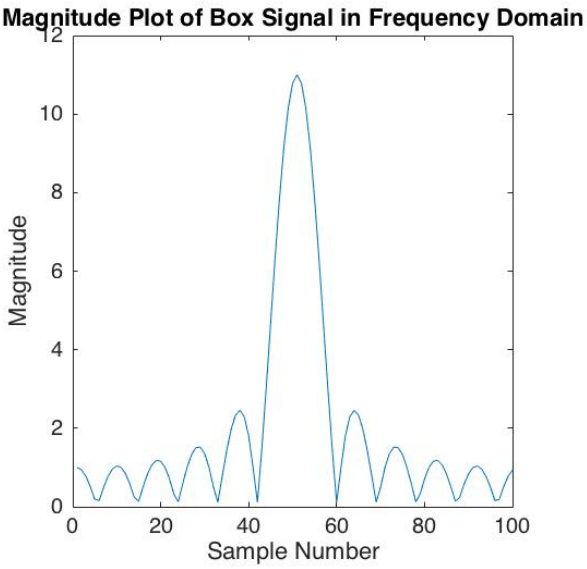
A box signal of width 5, and its magnitude plot in the frequency domain.



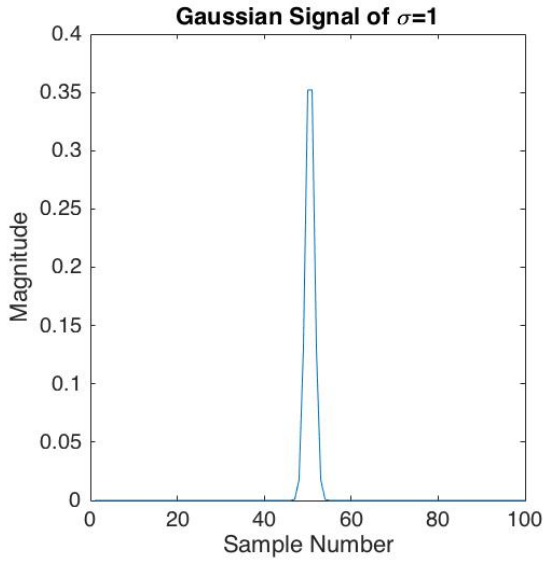


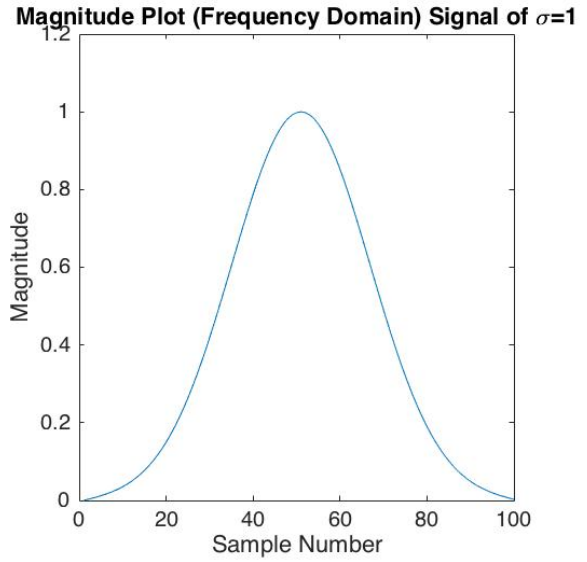
A box signal of width 10, and its magnitude plot in the frequency domain.



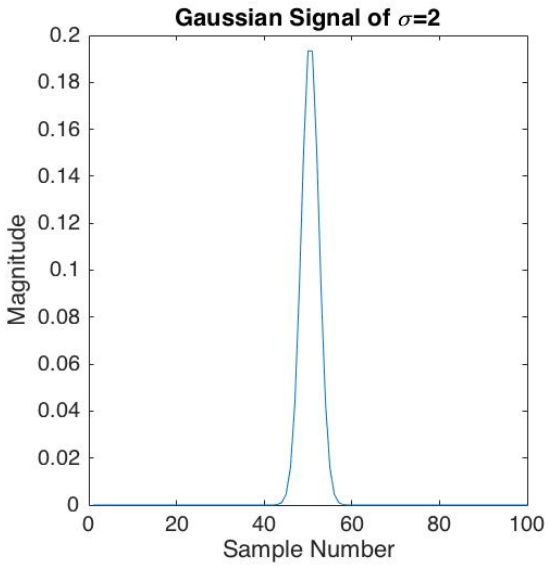


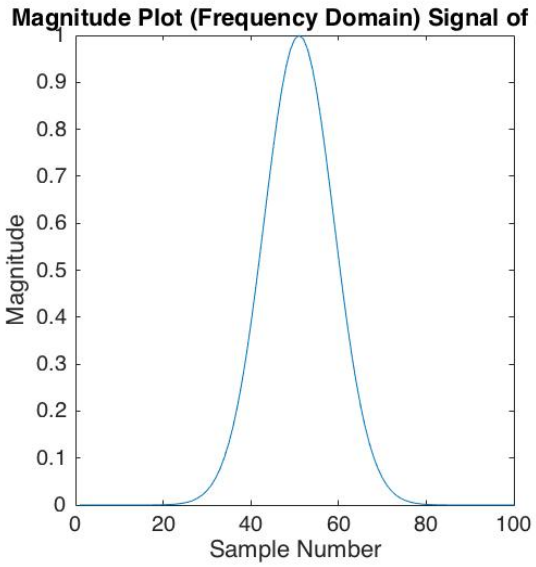
A Gaussian signal with sigma = 1, and its magnitude plot in the frequency domain.





A Gaussian signal with sigma = 2, and its magnitude plot in the frequency domain.





Combining the magnitude information and phase information of two different images, results in the following.

Image 1

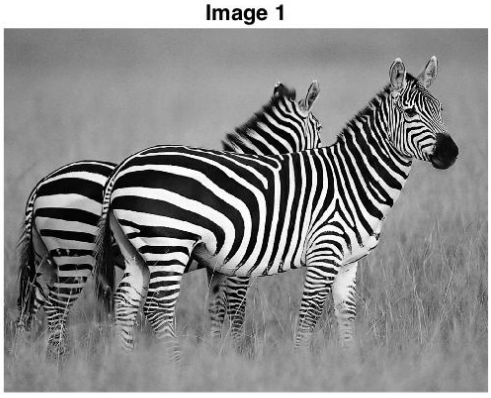
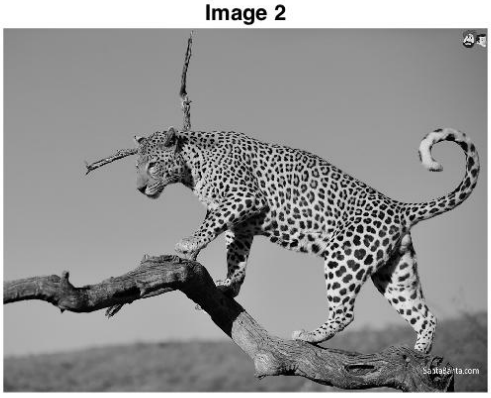
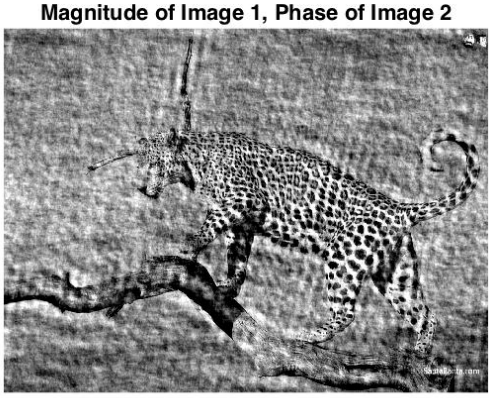


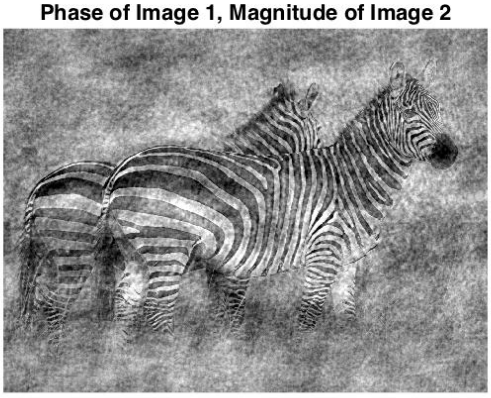
Image 2



Magnitude of Image 1, Phase of Image 2



Phase of Image 1, Magnitude of Image 2



7) Gradient Based Edge Detector

The gradient based edge detection was performed using the following pieces of code:

homework2_7.m

```
% Reading image and converting to grascale
img = im2double(rgb2gray(imread('img1.jpg')));

% Smoothing and finding the edges using a sigma value of 1 for gaussian
% smoothing before applying gradient derivative filter i.e. the Sobel
% operator.
[ edge_magnitude, edge_direction ] = smooth_find_edges(img, 1);
subplot(2,2,1);
imshow(edge_magnitude);
title('Edge Magnitude with \sigma=1');
subplot(2,2,2);
imshow(edge_direction);
title('Edge Orientation with \sigma=1');

% Smoothing and finding the edges using sigma value of 2 for gaussian
% smoothing before applying the gradient dericative filter i.e. the Sobel
% operator.
[ edge_magnitude, edge_direction ] = smooth_find_edges(img, 2);
subplot(2,2,3);
imshow(edge_magnitude);
title('Edge Magnitude with \sigma=2');
subplot(2,2,4);
imshow(edge_direction);
title('Edge Orientation with \sigma=2');
```

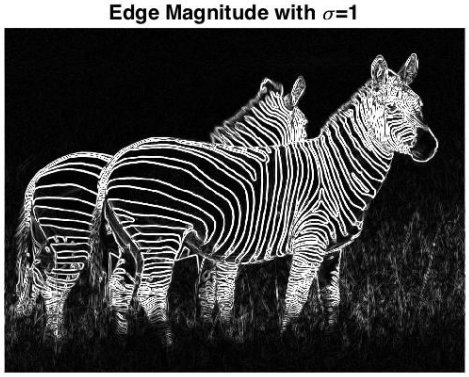
smooth_find_edges.m

```
% Function that applies Gaussian Smoothing using a sigma value; and then
% finds the directional derivative along the x and y direction using the
% Sobel operator, and uses the results to find the magnitude and direction
% of edge and return them.
function [ edge_magnitude, edge_direction ] = smooth_find_edges( img, sigma
)

    % Generating a Gaussian Filter of size 3x3
    % Wiht a sigma as specified by the input argument
    gaussian_filter = fspecial('gaussian', [5 5], sigma);

    % Smoothing image before finding horizontal and vertical derivatives
    img_smooth = conv2(img, gaussian_filter, 'same');

    % Using the Sobel operator to find the derivative in the
    % Horizontal and vertical direction
    sobel_y = [ -1 -2 -1; 0 0 0; 1 2 1 ];
    sobel_x = [ -1 0 1; -2 0 2; -1 0 1 ];
    img_dx = conv2(img_smooth, sobel_x, 'same');
    img_dy = conv2(img_smooth, sobel_y, 'same');

    % Finding the edge direction and edge magnitudes
    edge_magnitude = (img_dx.^2 + img_dy.^2).^(0.5);
    edge_direction = atan(img_dy./img_dx);
end
```

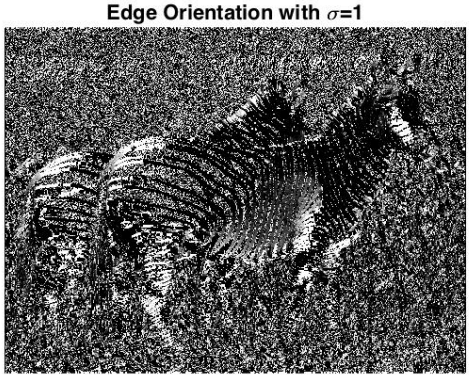
The gradient based edge detection was performed on the following image after conversion to grayscale.

The result of the edge detection, after performing Gaussian smoothing with sigma values, 1 and 2 respectively, resulted in the following edge magnitudes and direction. It is evident that, with increase in the standard deviation of the smoothing function, edge details in the image are lost.

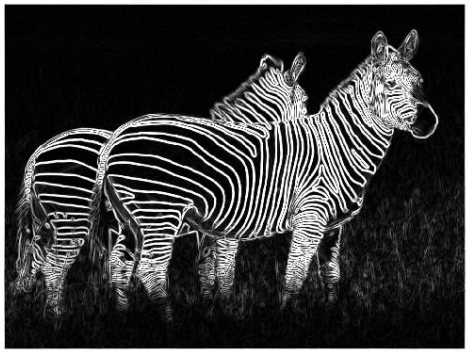
Edge Magnitude with σ=1



Edge Orientation with σ=1



Edge Magnitude with σ=2



Edge Orientation with σ=2

8) Correlation based Object Detector

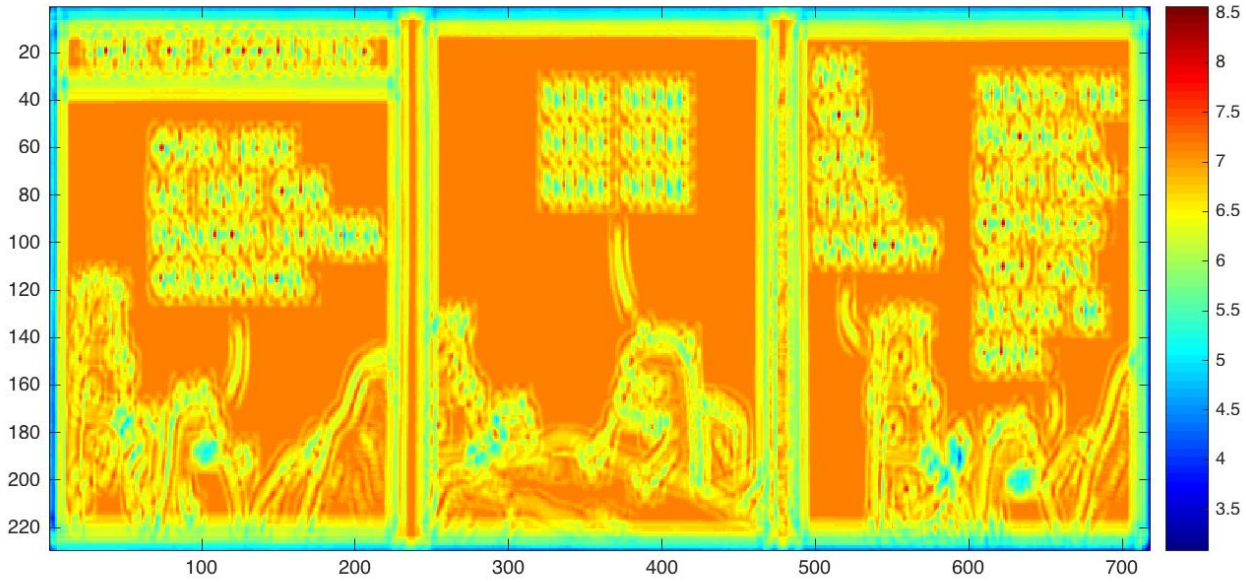
The following piece of code performs, object detection using correlation. It looks for a template in an image using cross-correlation.

homework2_8.m

```
% Loading the image and template

img = im2double(imread('dilbert.jpg'));
template = im2double(imread('dilbert-template2.jpg'));

% Performing normalized correlation and non maximal supression
correlation = conv2(img, rot90(template,2), 'same');
cgg = conv2(img.^2, ones(size(template)), 'same');
nc = correlation./cgg.^0.5;

% Displaying the Normalized Correlation values
nc = nc./max(max(nc));
imagesc(nc);
colorbar;
colormap jet;

% Performing non maximal supression
nc = non_maximal_supression(nc);

% Thresholding the correlation values
threshold = 0.95;
[ ypeaks, xpeaks ] = find(nc>threshold);

% Drawing squares around the pixels with high enough threshold
hFig = figure;
hAx  = axes;
imshow(img,'Parent', hAx);
hold on;
template_x = size(template,2)/2;
template_y = size(template,1)/2;
for i = 1:size(ypeaks, 1)
    ypeak = ypeaks(i,1);
    xpeak = xpeaks(i,1);
    % imrect(hAx, [xpeak - template_x, ypeak - template_y,
size(template,2), size(template,1)]);
    plot(xpeak, ypeak, 'o', 'MarkerSize', max(size(template)), 'Color',
'red');
end
```

The dilbert.jpg image was used, along with the following template of the letter O to do the object matching. The images are as shown below.

The visualization of the result of the cross correlation is as shown below.



The non-maximal suppression is as done in the ***non_maximal_supression.m*** file, as shown below.

```
function [ a ] = non_maximal_supression( a )
    for i = 2:size(a,1)-1
        for j = 2:size(a,2)-1
            x = a(i,j);
            if(a(i-1,j-1) > x || a(i-1,j) > x || a(i-1,j+1) > x || a(i,j-1)
> x...
                    || a(i,j+1) > x || a(i+1,j-1) > x || a(i+1,j) > x ||
a(i,j+1) > x)
                a(i,j) = 0;
            end
        end
    end
end
```

The final detection result was visualized by tweaking the threshold parameter and by plotting circles over the detections. The following image shows the final result.

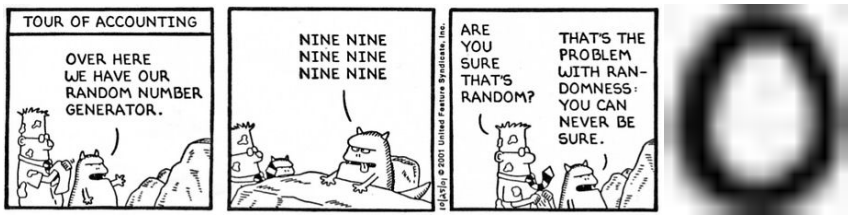
9) Sum of Squared Differences (SSD based Object Detection

The following piece of code performs, object detection using sum of squared differences. It looks for a template in an image using SSD.

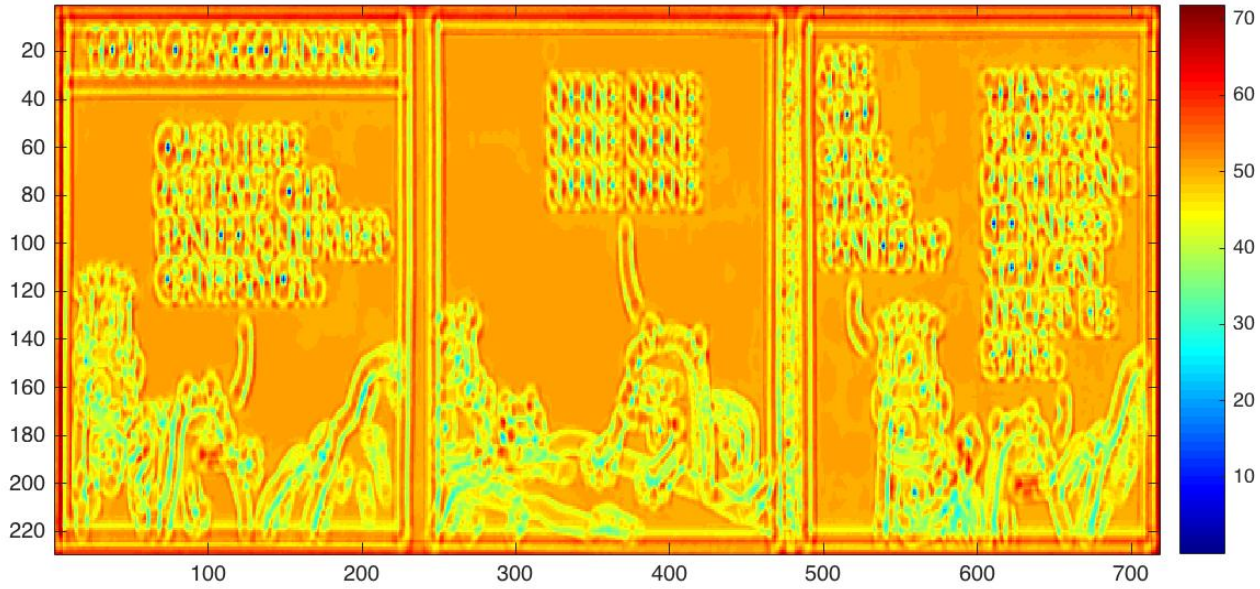
homework2_9.m

```
% Loading the image and template

img = im2double(imread('dilbert.jpg'));
template = im2double(imread('dilbert-template2.jpg'));

% Performing normalized correlation and non maximal supression
c = conv2(img,rot90(template,2), 'same');
template_square = sum(sum(template.^2));
image_square = conv2(img.^2,ones(size(template)), 'same');
ssd = image_square - 2*c + template_square;
% Displaying the Normalized Correlation values
imagesc(ssd);
colorbar;
colormap jet;

ssd = ssd./max(max(ssd));
ssd = non_minimal_supression(ssd);

% Thresholding the correlation values
threshold = 0.12; % 0.265;
% [ ypeaks, xpeaks ] = find(nc == min(min(nc)));
[ ypeaks, xpeaks ] = find(ssd<threshold);

% Drawing squares around the pixels with high enough threshold
hFig = figure;
hAx  = axes;
imshow(img,'Parent', hAx);
hold on;
plot(xpeaks,ypeaks, 'o', 'MarkerSize', max(size(template)), 'Color',
'blue');
```

The dilbert.jpg image was used, along with the following template of the letter O to do the object matching. The images are as shown below.



The visualization of the result of the sum of squared differences is as shown below.



The non-minimal suppression is as done in the ***non_minimall_supression.m*** file, as shown below.

```
function [ a ] = non_minimal_supression( a )
    for i = 2:size(a,1)-1
        for j = 2:size(a,2)-1
            x = a(i,j);
            if(a(i-1,j-1) < x || a(i-1,j) < x || a(i-1,j+1) < x || a(i,j-1)
< x...
                    || a(i,j+1) < x || a(i+1,j-1) < x || a(i+1,j) < x ||
a(i,j+1) < x)
                a(i,j) = 1;
            end
        end
    end
end
```

The final detection result was visualized by tweaking the threshold parameter and by plotting circles over the detections. The following image shows the final result.

Form my observations, SSD and cross-correlation both seem to work equally well. The only notable differences seem to be that, when looking for similarity, in SSD we look for the least distance, and in cross-correlation we look for the highest correlation factor. From the following examples tried with different templates and the ***dilbert.jpg*** the results from SSD and cross correlation are equally good.

The template used was the following.

The results from SSD and cross correlation are as given below.